Package 'policytree'

June 23, 2023

Title Policy Learning via Doubly Robust Empirical Welfare Maximization over Trees

Version 1.2.2

Description Learn optimal policies via doubly robust empirical welfare maximization over trees. Given doubly robust reward estimates, this package finds a rule-based treatment prescription policy, where the policy takes the form of a shallow decision tree that is globally (or close to) optimal.

Depends R (>= 3.5.0)

License GPL-3

Encoding UTF-8

Suggests testthat (>= 3.0.4), DiagrammeR

RoxygenNote 7.2.3

LinkingTo Rcpp, BH

Imports Rcpp, grf (>= 2.0.0)

URL https://github.com/grf-labs/policytree

BugReports https://github.com/grf-labs/policytree/issues

NeedsCompilation yes

Author Erik Sverdrup [aut, cre],
Ayush Kanodia [aut],
Zhengyuan Zhou [aut],
Susan Athey [aut],
Stefan Wager [aut]

Maintainer Erik Sverdrup <erikcs@stanford.edu>

Repository CRAN

Date/Publication 2023-06-23 04:10:02 UTC

R topics documented:

`conditional_means.causal_forest`
Estimate mean rewards μ for each treatment a

Description

$\mu_a = m(x) + (1 - e_a(x))\tau_a(x)$

Usage

```
## S3 method for class 'causal_forest'
conditional_means(object, ...)

## S3 method for class 'causal_survival_forest'
conditional_means(object, ...)

## S3 method for class 'instrumental_forest'
conditional_means(object, ...)

## S3 method for class 'multi_arm_causal_forest'
conditional_means(object, outcome = 1, ...)

conditional_means(object, ...)
```

Arguments

`object`	An appropriate causal forest type object
`...`	Additional arguments
`outcome`	Only used with multi arm causal forets. In the event the forest is trained with multiple outcomes Y, a column number/name specifying the outcome of interest. Default is 1.

Value

A matrix of estimated mean rewards

Methods (by class)

- `conditional_means(causal_forest)`: Mean rewards μ for control/treated
- `conditional_means(causal_survival_forest)`: Mean rewards μ for control/treated
- `conditional_means(instrumental_forest)`: Mean rewards μ for control/treated
- `conditional_means(multi_arm_causal_forest)`: Mean rewards μ for each treatment a

Examples

```
# Compute conditional means for a multi-arm causal forest
n <- 500
p <- 10
X <- matrix(rnorm(n * p), n, p)
W <- as.factor(sample(c("A", "B", "C"), n, replace = TRUE))
Y <- X[, 1] + X[, 2] * (W == "B") + X[, 3] * (W == "C") + runif(n)
forest <- grf::multi_arm_causal_forest(X, Y, W)
mu.hats <- conditional_means(forest)
head(mu.hats)

# Compute conditional means for a causal forest
n <- 500
p <- 10
X <- matrix(rnorm(n * p), n, p)
W <- rbinom(n, 1, 0.5)
Y <- pmax(X[, 1], 0) * W + X[, 2] + pmin(X[, 3], 0) + rnorm(n)
c.forest <- grf::causal_forest(X, Y, W)
mu.hats <- conditional_means(c.forest)
```

`double_robust_scores.causal_forest` *Matrix Γ of scores for each treatment a*

Description

Computes a matrix of double robust scores $\Gamma_{ia} = \mu_a(x) + \frac{1}{e_a(x)}(Y_i - \mu_a(x))1(A_i = a)$

Usage

```
## S3 method for class 'causal_forest'
double_robust_scores(object, ...)

## S3 method for class 'causal_survival_forest'
double_robust_scores(object, ...)

## S3 method for class 'instrumental_forest'
double_robust_scores(object, compliance.score = NULL, ...)
```

```
## S3 method for class 'multi_arm_causal_forest'
double_robust_scores(object, outcome = 1, ...)

double_robust_scores(object, ...)
```

Arguments

`object`	An appropriate causal forest type object
`...`	Additional arguments
`compliance.score`	An estimate of the causal effect of Z on W. i.e., Delta(X) = E(W | X, Z = 1) - E(W | X, Z = 0), for each sample i = 1, ..., n. If NULL (default) then this is estimated with a causal forest.
`outcome`	Only used with multi arm causal forets. In the event the forest is trained with multiple outcomes Y, a column number/name specifying the outcome of interest. Default is 1.

Details

This is the matrix used for CAIPWL (Cross-fitted Augmented Inverse Propensity Weighted Learning)

Value

A matrix of scores for each treatment

Methods (by class)

- `double_robust_scores(causal_forest)`: Scores (Γ_0, Γ_1)
- `double_robust_scores(causal_survival_forest)`: Scores (Γ_0, Γ_1)
- `double_robust_scores(instrumental_forest)`: Scores $(-\Gamma, \Gamma)$
- `double_robust_scores(multi_arm_causal_forest)`: Matrix Γ of scores for each treatment a

Note

For instrumental_forest this method returns $(-\Gamma_i, \Gamma_i)$ where Γ_i is the double robust estimator of the treatment effect as in eqn. (44) in Athey and Wager (2021).

References

Athey, Susan, and Stefan Wager. "Policy Learning With Observational Data." Econometrica 89.1 (2021): 133-161.

Examples

```
# Compute double robust scores for a multi-arm causal forest
n <- 500
p <- 10
X <- matrix(rnorm(n * p), n, p)
W <- as.factor(sample(c("A", "B", "C"), n, replace = TRUE))
Y <- X[, 1] + X[, 2] * (W == "B") + X[, 3] * (W == "C") + runif(n)
forest <- grf::multi_arm_causal_forest(X, Y, W)
scores <- double_robust_scores(forest)
head(scores)

# Compute double robust scores for a causal forest
n <- 500
p <- 10
X <- matrix(rnorm(n * p), n, p)
W <- rbinom(n, 1, 0.5)
Y <- pmax(X[, 1], 0) * W + X[, 2] + pmin(X[, 3], 0) + rnorm(n)
c.forest <- grf::causal_forest(X, Y, W)
scores <- double_robust_scores(c.forest)
```

---

gen_data_epl *Example data generating process from Policy Learning With Observational Data*

Description

The DGP from section 5.2 in Athey and Wager (2021)

Usage

```
gen_data_epl(n, type = c("continuous", "jump"))
```

Arguments

n	Number of observations
type	tau is "continuous" (default - equation 46) or exhibits "jumps" (equation 47)

Value

A list

References

Athey, Susan, and Stefan Wager. "Policy Learning With Observational Data." Econometrica 89.1 (2021): 133-161.

gen_data_mapl *Example data generating process from Offline Multi-Action Policy Learning: Generalization and Optimization*

Description

The DGP from section 6.4.1 in Zhou, Athey, and Wager (2023): There are $d = 3$ actions (a_0, a_1, a_2) which depend on 3 regions the covariates $X \sim U[0,1]^p$ reside in. Observed outcomes: $Y \sim N(\mu_{a_i}(X_i), 4)$

Usage

```
gen_data_mapl(n, p = 10, sigma2 = 4)
```

Arguments

n	Number of observations X.
p	Number of features (minimum 7). Default is 10.
sigma2	Noise variance. Default is 4.

Value

A list with realized action a_i, region r_i, conditional mean μ, outcome Y and covariates X

References

Zhou, Zhengyuan, Susan Athey, and Stefan Wager. "Offline multi-action policy learning: Generalization and optimization." Operations Research 71.1 (2023).

hybrid_policy_tree *Hybrid tree search*

Description

Finds a depth k tree by looking ahead l steps.

Usage

```
hybrid_policy_tree(
  X,
  Gamma,
  depth = 3,
  search.depth = 2,
  split.step = 1,
  min.node.size = 1,
  verbose = TRUE
)
```

Arguments

X	The covariates used. Dimension $N * p$ where p is the number of features.
Gamma	The rewards for each action. Dimension $N * d$ where d is the number of actions.
depth	The depth of the fitted tree. Default is 3.
search.depth	Depth to look ahead when splitting. Default is 2.
split.step	An optional approximation parameter, the number of possible splits to consider when performing tree search. split.step = 1 (default) considers every possible split, split.step = 10 considers splitting at every 10'th sample and may yield a substantial speedup for dense features. Manually rounding or re-encoding continuous covariates with very high cardinality in a problem specific manner allows for finer-grained control of the accuracy/runtime tradeoff and may in some cases be the preferred approach.
min.node.size	An integer indicating the smallest terminal node size permitted. Default is 1.
verbose	Give verbose output. Default is TRUE.

Details

Builds deeper trees by iteratively using exact tree search to look ahead l splits. For example, with depth = 3 and search.depth = 2, the root split is determined by a depth 2 exact tree, and two new depth 2 trees are fit in the two immediate children using exact tree search, leading to a total depth of 3 (the resulting tree may be shallower than the specified depth depending on whether leaf nodes were pruned or not). This algorithm scales with some coefficient multiple of the runtime of a search.depth policy_tree, which means that for this approach to be feasible it needs an (n, p, d) configuration in which a search.depth policy_tree runs in reasonable time.

The algorithm: desired depth is given by depth. Each node is split using exact tree search with depth = search.depth. When we reach a node where the current level + search.depth is equal to depth, we stop and attach the search.depth subtree to this node. We also stop if the best search.depth split yielded a leaf node.

Value

A policy_tree object.

Examples

```
# Fit a depth three tree on doubly robust treatment effect estimates from a causal forest.
n <- 1500
p <- 5
X <- round(matrix(rnorm(n * p), n, p), 2)
W <- rbinom(n, 1, 1 / (1 + exp(X[, 3])))
tau <- 1 / (1 + exp((X[, 1] + X[, 2]) / 2)) - 0.5
Y <- X[, 3] + W * tau + rnorm(n)
c.forest <- grf::causal_forest(X, Y, W)
dr.scores <- double_robust_scores(c.forest)

tree <- hybrid_policy_tree(X, dr.scores, depth = 3)
```

```
# Predict treatment assignment.
predicted <- predict(tree, X)
```

`multi_causal_forest` *(deprecated) One vs. all causal forest for multiple treatment effect estimation*

Description

Since policytree version 1.1 this function is deprecated in favor of the new estimator `multi_arm_causal_forest` available in GRF (version 2+). This function will continue to work for now but passes its arguments onto the "conformable" `multi_arm_causal_forest` in GRF, with a warning. (Note: for policy learning this forest works as before, but for individual point predictions, they differ as `multi_arm_causal_forest` predicts contrasts. See the GRF documentation example for details.)

Usage

```
multi_causal_forest(
  X,
  Y,
  W,
  Y.hat = NULL,
  W.hat = NULL,
  num.trees = 2000,
  sample.weights = NULL,
  clusters = NULL,
  equalize.cluster.weights = FALSE,
  sample.fraction = 0.5,
  mtry = min(ceiling(sqrt(ncol(X)) + 20), ncol(X)),
  min.node.size = 5,
  honesty = TRUE,
  honesty.fraction = 0.5,
  honesty.prune.leaves = TRUE,
  alpha = 0.05,
  imbalance.penalty = 0,
  stabilize.splits = TRUE,
  ci.group.size = 2,
  tune.parameters = "none",
  tune.num.trees = 200,
  tune.num.reps = 50,
  tune.num.draws = 1000,
  compute.oob.predictions = TRUE,
  orthog.boosting = FALSE,
  num.threads = NULL,
  seed = runif(1, 0, .Machine$integer.max)
)
```

Arguments

X	The covariates used in the causal regression.
Y	The outcome (must be a numeric vector with no NAs).
W	The treatment assignment (must be a categorical vector with no NAs).
Y.hat	Estimates of the expected responses E[Y | Xi], marginalizing over treatment. If Y.hat = NULL, these are estimated using a separate regression forest. See section 6.1.1 of the GRF paper for further discussion of this quantity. Default is NULL.
W.hat	Matrix with estimates of the treatment propensities E[Wk | Xi]. If W.hat = NULL, these are estimated using a k separate regression forests. Default is NULL.
num.trees	Number of trees grown in the forest. Note: Getting accurate confidence intervals generally requires more trees than getting accurate predictions. Default is 2000.
sample.weights	(experimental) Weights given to each sample in estimation. If NULL, each observation receives the same weight. Note: To avoid introducing confounding, weights should be independent of the potential outcomes given X. Default is NULL.
clusters	Vector of integers or factors specifying which cluster each observation corresponds to. Default is NULL (ignored).
equalize.cluster.weights	If FALSE, each unit is given the same weight (so that bigger clusters get more weight). If TRUE, each cluster is given equal weight in the forest. In this case, during training, each tree uses the same number of observations from each drawn cluster: If the smallest cluster has K units, then when we sample a cluster during training, we only give a random K elements of the cluster to the tree-growing procedure. When estimating average treatment effects, each observation is given weight 1/cluster size, so that the total weight of each cluster is the same. Note that, if this argument is FALSE, sample weights may also be directly adjusted via the sample.weights argument. If this argument is TRUE, sample.weights must be set to NULL. Default is FALSE.
sample.fraction	Fraction of the data used to build each tree. Note: If honesty = TRUE, these subsamples will further be cut by a factor of honesty.fraction. Default is 0.5.
mtry	Number of variables tried for each split. Default is $\sqrt{p} + 20$ where p is the number of variables.
min.node.size	A target for the minimum number of observations in each tree leaf. Note that nodes with size smaller than min.node.size can occur, as in the original randomForest package. Default is 5.
honesty	Whether to use honest splitting (i.e., sub-sample splitting). Default is TRUE. For a detailed description of honesty, honesty.fraction, honesty.prune.leaves, and recommendations for parameter tuning, see the grf algorithm reference.
honesty.fraction	The fraction of data that will be used for determining splits if honesty = TRUE. Corresponds to set J1 in the notation of the paper. Default is 0.5 (i.e. half of the data is used for determining splits).

`honesty.prune.leaves`
: If true, prunes the estimation sample tree such that no leaves are empty. If false, keep the same tree as determined in the splits sample (if an empty leave is encountered, that tree is skipped and does not contribute to the estimate). Setting this to false may improve performance on small/marginally powered data, but requires more trees (note: tuning does not adjust the number of trees). Only applies if honesty is enabled. Default is TRUE.

`alpha`
: A tuning parameter that controls the maximum imbalance of a split. Default is 0.05.

`imbalance.penalty`
: A tuning parameter that controls how harshly imbalanced splits are penalized. Default is 0.

`stabilize.splits`
: Whether or not the treatment should be taken into account when determining the imbalance of a split. Default is TRUE.

`ci.group.size`
: The forest will grow ci.group.size trees on each subsample. In order to provide confidence intervals, ci.group.size must be at least 2. Default is 2.

`tune.parameters`
: A vector of parameter names to tune. If "all": all tunable parameters are tuned by cross-validation. The following parameters are tunable: ("sample.fraction", "mtry", "min.node.size", "honesty.fraction", "honesty.prune.leaves", "alpha", "imbalance.penalty"). If honesty is false these parameters are not tuned. Default is "none" (no parameters are tuned).

`tune.num.trees`
: The number of trees in each 'mini forest' used to fit the tuning model. Default is 200.

`tune.num.reps`
: The number of forests used to fit the tuning model. Default is 50.

`tune.num.draws`
: The number of random parameter values considered when using the model to select the optimal parameters. Default is 1000.

`compute.oob.predictions`
: Whether OOB predictions on training set should be precomputed. Default is TRUE.

`orthog.boosting`
: Deprecated and unused after version 1.0.4.

`num.threads`
: Number of threads used in training. By default, the number of threads is set to the maximum hardware concurrency.

`seed`
: The seed of the C++ random number generator.

Value

A warning will be issued and this function passes its arguments onto the new estimator `multi_arm_causal_forest` and returns that object.

plot.policy_tree *Plot a policy_tree tree object.*

Description

Plot a policy_tree tree object.

Usage

```
## S3 method for class 'policy_tree'
plot(x, leaf.labels = NULL, ...)
```

Arguments

x	The tree to plot.
leaf.labels	An optional character vector of leaf labels for each treatment.
...	Additional arguments (currently ignored).

Examples

```
# Plot a policy_tree object
## Not run:
n <- 250
p <- 10
X <- matrix(rnorm(n * p), n, p)
W <- as.factor(sample(c("A", "B", "C"), n, replace = TRUE))
Y <- X[, 1] + X[, 2] * (W == "B") + X[, 3] * (W == "C") + runif(n)
multi.forest <- grf::multi_arm_causal_forest(X = X, Y = Y, W = W)
Gamma.matrix <- double_robust_scores(multi.forest)
tree <- policy_tree(X, Gamma.matrix, depth = 2)
plot(tree)

# Provide optional names for the treatment names in each leaf node
# `action.names` is by default the column names of the reward matrix
plot(tree, leaf.labels = tree$action.names)
# Providing a custom character vector
plot(tree, leaf.labels = c("treatment A", "treatment B", "placebo C"))

# Saving a plot in a vectorized SVG format can be done with the `DiagrammeRsvg` package.
install.packages("DiagrammeRsvg")
tree.plot = plot(tree)
cat(DiagrammeRsvg::export_svg(tree.plot), file = 'plot.svg')

## End(Not run)
```

`policy_tree` *Fit a policy with exact tree search*

Description

Finds the optimal (maximizing the sum of rewards) depth k tree by exhaustive search. If the optimal action is the same in both the left and right leaf of a node, the node is pruned.

Usage

```
policy_tree(
  X,
  Gamma,
  depth = 2,
  split.step = 1,
  min.node.size = 1,
  verbose = TRUE
)
```

Arguments

`X`	The covariates used. Dimension $N * p$ where p is the number of features.
`Gamma`	The rewards for each action. Dimension $N * d$ where d is the number of actions.
`depth`	The depth of the fitted tree. Default is 2.
`split.step`	An optional approximation parameter, the number of possible splits to consider when performing tree search. split.step = 1 (default) considers every possible split, split.step = 10 considers splitting at every 10'th sample and may yield a substantial speedup for dense features. Manually rounding or re-encoding continuous covariates with very high cardinality in a problem specific manner allows for finer-grained control of the accuracy/runtime tradeoff and may in some cases be the preferred approach.
`min.node.size`	An integer indicating the smallest terminal node size permitted. Default is 1.
`verbose`	Give verbose output. Default is TRUE.

Details

Exact tree search is intended as a way to find shallow (i.e. depth 2 or 3) globally optimal tree-based polices on datasets of "moderate" size. The amortized runtime of exact tree search is $O(p^k n^k (logn + d) + pnlogn)$ where p is the number of features, n the number of distinct observations, d the number of treatments, and k >= 1 the tree depth. Due to the exponents in this expression, exact tree search will not scale to datasets of arbitrary size.

As an example, the runtime of a depth two tree scales quadratically with the number of observations, implying that doubling the number of samples will quadruple the runtime. n refers to the number of distinct observations, substantial speedups can be gained when the features are discrete (with all binary features, the runtime will be ~ linear in n), and it is therefore beneficial to round down/re-encode very dense data to a lower cardinality (the optional parameter `split.step` emulates this, though rounding/re-encoding allow for finer-grained control).

Value

A policy_tree object.

References

Athey, Susan, and Stefan Wager. "Policy Learning With Observational Data." Econometrica 89.1 (2021): 133-161.

Sverdrup, Erik, Ayush Kanodia, Zhengyuan Zhou, Susan Athey, and Stefan Wager. "policytree: Policy learning via doubly robust empirical welfare maximization over trees." Journal of Open Source Software 5, no. 50 (2020): 2232.

Zhou, Zhengyuan, Susan Athey, and Stefan Wager. "Offline multi-action policy learning: Generalization and optimization." Operations Research 71.1 (2023).

See Also

hybrid_policy_tree for building deeper trees.

Examples

```
# Construct doubly robust scores using a causal forest.
n <- 10000
p <- 10
# Discretizing continuous covariates decreases runtime for policy learning.
X <- round(matrix(rnorm(n * p), n, p), 2)
colnames(X) <- make.names(1:p)
W <- rbinom(n, 1, 1 / (1 + exp(X[, 3])))
tau <- 1 / (1 + exp((X[, 1] + X[, 2]) / 2)) - 0.5
Y <- X[, 3] + W * tau + rnorm(n)
c.forest <- grf::causal_forest(X, Y, W)

# Retrieve doubly robust scores.
dr.scores <- double_robust_scores(c.forest)

# Learn a depth-2 tree on a training set.
train <- sample(1:n, n / 2)
tree <- policy_tree(X[train, ], dr.scores[train, ], 2)
tree

# Evaluate the tree on a test set.
test <- -train

# One way to assess the policy is to see whether the leaf node (group) the test set samples
# are predicted to belong to have mean outcomes in accordance with the prescribed policy.

# Get the leaf node assigned to each test sample.
node.id <- predict(tree, X[test, ], type = "node.id")

# Doubly robust estimates of E[Y(control)] and E[Y(treated)] by leaf node.
values <- aggregate(dr.scores[test, ], by = list(leaf.node = node.id),
                    FUN = function(dr) c(mean = mean(dr), se = sd(dr) / sqrt(length(dr))))
```

```
print(values, digits = 1)

# Take cost of treatment into account by, for example, offsetting the objective
# with an estimate of the average treatment effect.
ate <- grf::average_treatment_effect(c.forest)
cost.offset <- ate[["estimate"]]
dr.scores[, "treated"] <- dr.scores[, "treated"] - cost.offset
tree.cost <- policy_tree(X, dr.scores, 2)

# Predict treatment assignment for each sample.
predicted <- predict(tree, X)

# If there are too many covariates to make tree search computationally feasible, then one
# approach is to consider for example only the top features according to GRF's variable importance.
var.imp <- grf::variable_importance(c.forest)
top.5 <- order(var.imp, decreasing = TRUE)[1:5]
tree.top5 <- policy_tree(X[, top.5], dr.scores, 2, split.step = 50)
```

`predict.policy_tree` *Predict method for policy_tree*

Description

Predict values based on fitted policy_tree object.

Usage

```
## S3 method for class 'policy_tree'
predict(object, newdata, type = c("action.id", "node.id"), ...)
```

Arguments

`object`	policy_tree object
`newdata`	Points at which predictions should be made. Note that this matrix should have the same number of columns as the training matrix, and that the columns must appear in the same order.
`type`	The type of prediction required, "action.id" is the action id and "node.id" is the integer id of the leaf node the sample falls into. Default is "action.id".
`...`	Additional arguments (currently ignored).

Value

A vector of predictions. For type = "action.id" each element is an integer from 1 to d where d is the number of columns in the reward matrix. For type = "node.id" each element is an integer corresponding to the node the sample falls into (level-ordered).

Examples

```
# Construct doubly robust scores using a causal forest.
n <- 10000
p <- 10
# Discretizing continuous covariates decreases runtime for policy learning.
X <- round(matrix(rnorm(n * p), n, p), 2)
colnames(X) <- make.names(1:p)
W <- rbinom(n, 1, 1 / (1 + exp(X[, 3])))
tau <- 1 / (1 + exp((X[, 1] + X[, 2]) / 2)) - 0.5
Y <- X[, 3] + W * tau + rnorm(n)
c.forest <- grf::causal_forest(X, Y, W)

# Retrieve doubly robust scores.
dr.scores <- double_robust_scores(c.forest)

# Learn a depth-2 tree on a training set.
train <- sample(1:n, n / 2)
tree <- policy_tree(X[train, ], dr.scores[train, ], 2)
tree

# Evaluate the tree on a test set.
test <- -train

# One way to assess the policy is to see whether the leaf node (group) the test set samples
# are predicted to belong to have mean outcomes in accordance with the prescribed policy.

# Get the leaf node assigned to each test sample.
node.id <- predict(tree, X[test, ], type = "node.id")

# Doubly robust estimates of E[Y(control)] and E[Y(treated)] by leaf node.
values <- aggregate(dr.scores[test, ], by = list(leaf.node = node.id),
                    FUN = function(dr) c(mean = mean(dr), se = sd(dr) / sqrt(length(dr))))
print(values, digits = 1)

# Take cost of treatment into account by, for example, offsetting the objective
# with an estimate of the average treatment effect.
ate <- grf::average_treatment_effect(c.forest)
cost.offset <- ate[["estimate"]]
dr.scores[, "treated"] <- dr.scores[, "treated"] - cost.offset
tree.cost <- policy_tree(X, dr.scores, 2)

# Predict treatment assignment for each sample.
predicted <- predict(tree, X)

# If there are too many covariates to make tree search computationally feasible, then one
# approach is to consider for example only the top features according to GRF's variable importance.
var.imp <- grf::variable_importance(c.forest)
top.5 <- order(var.imp, decreasing = TRUE)[1:5]
tree.top5 <- policy_tree(X[, top.5], dr.scores, 2, split.step = 50)
```

`print.policy_tree` *Print a policy_tree object.*

Description

Print a policy_tree object.

Usage

```
## S3 method for class 'policy_tree'
print(x, ...)
```

Arguments

`x`	The tree to print.
`...`	Additional arguments (currently ignored).

Index